Title	Knight-Tourのプログラム (計算機によるゲームとパズルをめぐる諸問題研究会報告集)
Author(s)	筧, 捷彦
Citation	数理解析研究所講究録 (1970), 98: 57-67
Issue Date	1970-09
URL	http://hdl.handle.net/2433/108208
Right	
Type	Departmental Bulletin Paper
Textversion	publisher

数理解析研究所講究録
98巻 1970年 57-67

Knight-tour のプログラム

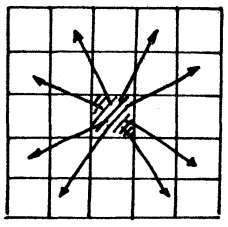
東大 計数工学科 筧 捷彦

§1. 序

パズルを計算機で解こうという場合の多くは、理論によって裏付けられてうまい方法があるわけではなくて、結局しらみつぶしに場合を尽くして解を求めることに落ちつくようである。あとはいかに無駄な場合の調査を省くかに工夫を求めることになる。

Knight-tour の問題に興味をもったのは、知人のみやげにケンブリッジのTSSのプログラムを貰ったのがきっかけであった。BCPLで書かれたそのプログラムを解読して、このパズルの存在を知り、しばらくの間取組んでみた顛末を報告する。

§2. パズル



Chess の Knight の動き方は他の駒に比べ特異であり、この動き方に基づいたパズルはいくつか考えられるが、その一つが Knight-tour である。

与えられた盤上で、各マス目をただ一度ずつ通過して全部のマス目を動くKnightの道を見出すこと。

n×nの盤について調べてみると、n=2, 3, 4の時は解が存在しない。n=5, 6では解が存在する事がわかった。またn=8（chess board）にも解は存在する。

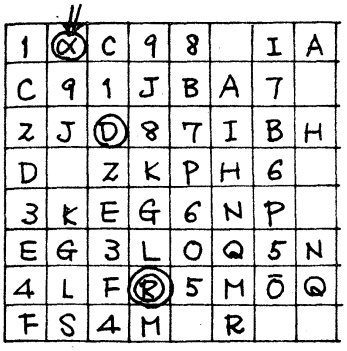
§3. しらみつぶし

人手で調べて解の存在がわかったので、先ずは計算機で唯一つでもよいから解を出させてみようと、「しらみつぶし」方法でプログラムを組んでみた結果が下図である。（n=8）

Chess boardのように白黒塗りわけると、Knightは必ず異る色のマス目へ動くから、高々32個の区別がつけばn=8の時の通過点の番号はわかる。1～9A～Wを使って32進ダンプしたのが下図で、0は未通過点を表わす。

```
10000000 00100000 20000000 00200000 30000000 00300000 40000000 00000000
10008000 00100070 20007000 00200060 30006000 00300050 40005000 00400000
10C980DA 0910BA70 20087UB0 00200060 30006000 00300050 40005000 00400000
10C980DA C910BA70 2UD870B0 D0200060 30E06000 EG300050 40F05000 F0400000
10C980IA C91JBA70 2JD87IBH D02K0H60 30EG6000 EG300050 40F05000 F0400000
10C980IA C91JBA70 2JD87IBH D02K0H60 3KEG6N00 EG3L005N 4LF05M00 F04M0000
10C980IA C91JBA70 2JD87IBH D02KPH60 3KEG6NP0 EG3LOQ5N 4LFR5MOQ FS4M0R00
```

第7行目をマス目に書き入れてみると、Knightは尺迄動いてはいるが、Ⓓの所へ進んだところで地点αを孤立させてしまっているので、既に道を誤っていることになる。東大大型センタで30秒走らせた時、この誤った道のしらみつぶしで

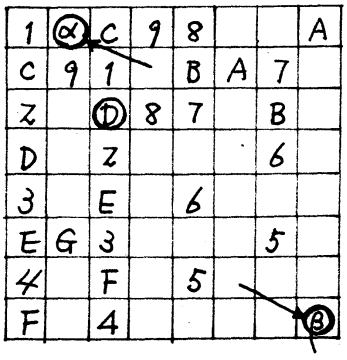


やっとJの所まで戻って来たところである。これではよほどの事が無いかぎりは唯一つの解に達することさえできそうにもない。

§4. Feasible test

そこで、こうした孤立点が生じないように、すべての未通過点に現在地からKnightの動きで行きつくことができることを確めながらstepを進めることにした。たまたま倍長のレジスタを使うと64 bitsで各bitに各々マス目を対応させることができるから、上の条件を満すか否かを調べるプログラムをアセンブラで組込んで走らせてみたのが下図である。

```
10000000 00100000 20000000 00200000 30000000 00300000 40000000 00400000
10000000 00100000 20000000 00200000 30005000 00300050 40005000 00400000
10008000 00100070 20007000 00200060 30005000 00300050 40005000 00400000
10098000 09100A70 20987000 00200060 30005000 00300050 40005000 00400000
10C98000 0910BA70 20B87000 00200060 30005000 00300050 40005000 00400000
10C9800A C910BA70 20D870B0 D0200060 30505000 00300050 40005000 00400000
10C9800A C910BA70 20D870B0 D0200060 30505000 EG300050 40F05000 FH400000
```



同じく第7行目をマス目に記入してみると左図のようになる。孤立点は無くなったが、Ⓐ, ③の2点が共に端点の形に強制されていて、このままstepを進めることが無意味であることがわかる。③が端

点に強制されたのは'5'を選んだ時であるから、Ⓓの次には、Ⓐが端点として残らぬようにⒶ自身を次の通過点として選ばねばならなかったことがわかる。同じく30秒走らせた時、やっとE点の選択変更点戻ってきたところであった。

§3,4の方法から得られた結果を反省してみると

- 誤った道に入ったことをなるべく早く検出すること。

  D stepで既に誤った時は各step 8方向あるとして $Z^{3\times(64-28)}$ $= Z^{108} \fallingdotseq 10^{30}$ 回ムダな調査を進めることになる。(実際はもっと少いが)

- feasible test にしても孤立点が生じる可能性は、1step進めたとき、そのマス目からknightの動きの関係にある高々8個の点であるから、前回の状態を上手に覚えておけば、かなりの手間が省ける。

§5. Algorithm

各マス目を $\langle x, y \rangle$, $\langle x, y \rangle$ からknightが動きうるマス目の数を l_{xy} としよう。knightが次に動く点として $\langle x, y \rangle$ を選びうる条件は次の様になる。

1. $\langle x, y \rangle$ は現地点から桂馬の関係にあり (1-1)
   盤内かつ未通過点であること。 (1-2)
   現地点につき(2-3)の条件があるときは
   それを満す点であること。 (1-3)
2. $\langle x, y \rangle$ から桂馬の関係にある点を $\langle x^*, y^* \rangle$ として

$l_{x^*y^*} := l_{x^*y^*} - 1$ とする。

- $l_{x^*y^*} = 0$ なる $\langle x^*, y^* \rangle$ が存在するとき

  $\langle x, y \rangle \dashrightarrow \langle x^*, y^* \rangle$ ‖ 行止まり

  $\langle x, y \rangle$ が tour の最終点の1つ手前の時のみ可。(この時解に達する) 他は不可 (2-1)

- $l_{x^*y^*} = 1$ なる $\langle x^*, y^* \rangle$ が存在するとき

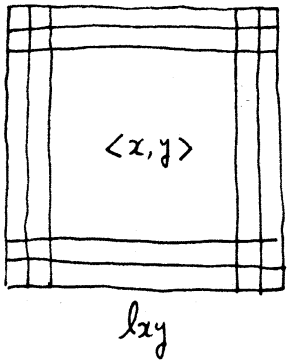
  $\langle x, y \rangle \dashrightarrow \langle x^*, y^* \rangle$ 1本道

  [$\langle x, y \rangle$ の次点として $\langle x^*, y^* \rangle$ を選ばない限り $\langle x^*, y^* \rangle$ は端点となる。]

  - ($l_{uv} = 1$ なる点の数) ≥ 3 不可 (2-2)
  - ($l_{uv} = 1$ なる点の数) $= 2$

    $\langle x, y \rangle$ の次の点として $l_{x^*y^*} = 1$ なる点 $\langle x^*, y^* \rangle$ の1つを選ぶ (2-3)

プログラムとしては、条件(1-2)も条件(2)と同等にしらべられる様に、与えられた盤よりひまわり大きな配列として l_{xy} を用意し、通過点及び盤外点に対応する l_{xy} の値を負にとるようにしてある。



この algorithm を用いてプログラムを作り、第1の点として $<1,1>$ を与え、$l_{11}<0$, $l_{i^*1^*} := l_{i^*1^*}-1$ に初期値を設定して解かせてみたところ、10秒（印刷枚数の方で打切られた）で98個の解がみつかった。丁度第1解の34番目より先だけを変更して得られる解を尽くしているところで打切られていた。この調子で解が存在するとすれば、$<1,1>$ を第1点とするものだけで（各stepでそれぞれ2つずつ解に達する選び方があるとして）$2^{34}\times 100 \fallingdotseq 10^{12}$, 更に第1点の選び方が対称性を考慮しても16通りあるから、気の遠くなるような数になる。

§6. Variation

そこで解の形に特徴のあるものを求めてみることにした。

a. 閉じた道

§5の algorithm を用いて、初期値設定を変える。即ち、第1の点として $<1,1>$ を与え、l_{11}, $l_{i^*1^*}$ は変更しないままにしておく。$l_{11}=2$ であることを利用して、条件として $l_{uv}=1$ なる点の数が1の時（これは第2点を選んだ時にのみ生じる）は、$<u,v>$ を次の点として選んではならぬ (2-4) を付加することで閉じた道を探し出すことができる。図-1に第1番目に見つかった解を示す。30秒で600個の解がみつかり、29番目より先だけを変更して得られる解が577個

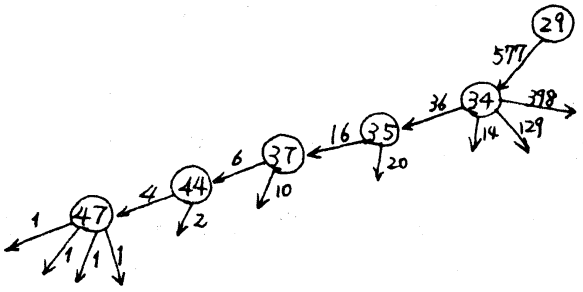
あった。

ANSWER(64)

1	8	11	22	3	18	13	16
10	23	2	7	12	15	4	19
55	64	9	24	21	6	17	14
30	35	54	63	40	25	20	5
47	56	29	36	53	62	41	26
34	31	46	39	28	37	52	61
57	48	33	44	59	50	27	42
32	45	58	49	38	43	60	51

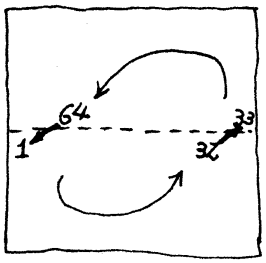
図-1

解の存在状況は下の様であった。

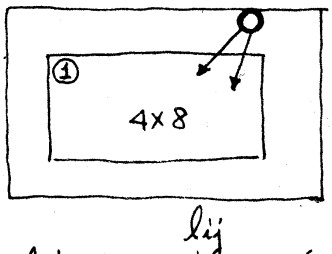


b. 点対称な解

閉じた道もまたあまりに多くの解が存在するので更に条件を付加してみた。盤の半分だけをまずうめつくし，次に残りの半分をうめつくす解で，閉じた道でありかつ，中心に対して対称な点の通過番号の差が32であるようなものを求めてみた。



これも同じ algorithm を用いて行ってみた。解に対する制約から，4×8 の盤を考えて，第1点を与えた

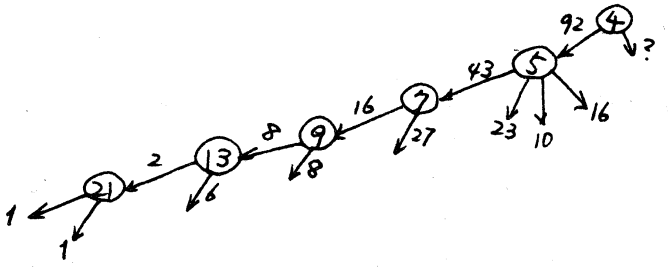


時、その対称点に対応する l_{ij} を(盤外)あたかもそこへも動ける様に与え(従って $l_{i^*j^*}$ も変化させておく)る。条件としてこの l_{ij} へ行かぬ様にする。(2-4') こうしてホめた第1解は下の様なものである。

1 TH ANSWER AT 5/24

①	22	1	28	9	32	13	18
6	25	4	21	14	17	12	31
23	2	27	8	29	10	19	16
26	5	24	3	20	15	30	11

第1点を隅に与えた時、30秒で丁度100個の解がホまった。第4点以後を変化させた解の個数は92個。存在状況は次の様であった。



§7. まとめ

始めは解があるか否か確めよう位のつもりでとりかかったが、あまりに解の数が多すぎて興味をそがれてしまった。一度 8×8 の解の総数を計算してみたいとは思っているが、今

のままの algorithm では計算時間があまりにかかりすぎるので更に工夫を要する。

今後は、「人間が学習し」ながら algorithm を作り出してゆくのではなく、「計算機自身に学習させ」るプログラムを考えてみたいと思っている。そうなってこそ計算機によるパズル解きの面白みもでてくるのではなかろうか。

付録として §6.b に用いたプログラムの この algorithm 部分をつけておいた。

```
      SUBROUTINE KNIGHT
      COMMON /P/ ENT,X,Y,D,IS
      COMMON /Q/ C,IC,IT,SC
      COMMON /R/ B,L,DX,DY
      COMMON /S/ SP,MAX,SX,SY,SD,SL,stack
      INTEGER ENT,X,Y,D
      INTEGER C
       INTEGER SC(64)
      INTEGER B(8,8),L(12,12),DX(8),DY(8)
      INTEGER SP,SX(64),SY(64),SD(64),SL(64)
      GO TO (10,20,30 ),ENT
   10 D=1
      B(X,Y)=SP
      IT=IT+1
   11 X=SX(SP)+DX(D)
      Y=SY(SP)+DY(D)
      LL=L(X+2,Y+2)
      IF(LL.LT.0) GO TO 13
      IF( Y.LE.0 ) GO TO 13
      IF( IS.EQ.2 .AND. LL.NE.1 ) GO TO 13
      NF=0
      DO 12 I=1,8
      M=X+DX(I)
      N=Y+DY(I)
      K=L(M+2,N+2)-1
      L(M+2,N+2)=K
      IF( K.EQ.0 ) NF=NF+1
      IF( K.EQ.1 ) IS=IS+1
   12 CONTINUE
      IF( LL.EQ.1 ) IS=IS-1
      IF( NF.NE.0 ) GO TO 32
      IF( IS.GT.2 ) GO TO 30
      SD(SP)=D
      SP=SP+1
      SX(SP)=X
      SY(SP)=Y
      SL(SP)=LL
      SC(SP)=0
      L(X+2,Y+2)=-1
      GO TO 10
   13 IF( D.GE.8 ) GO TO 14
      D=D+1
      GO TO 11
   14 IF( SP.LT.2 ) GO TO 33
      X=SX(SP)
      Y=SY(SP)
      IC=SC(SP)
      LL=SL(SP)
      L(X+2,Y+2)=LL
      ENT=2
      RETURN
   20 SP=SP-1
      D=SD(SP)
      SC(SP)=SC(SP)+IC
   30 IF( LL.EQ.1 ) IS=IS+1
      DO 31 I=1,8
      M=X+DX(I)
      N=Y+DY(I)
```

IS: 脚の count
IT: trial の count
SC: 脚の分布をしまう
L: lxy
SX, SY: <x, y>
SP: movement の番号 (SP) を入れる
DX, DY: knight の動き方 を入れてある
10: 第1の方向 I=set
<x, y> ← movement の番号
11: 次の点の候補 <x', y'> (1-1)
13: 盤面点, 盤外 (1-2)
盤 外 (2-4')
(1-3)
lx'*y'* := lx'*y'*-1
(2-1) → 32
(2-2) → 30
<x, y> からの方向 push-down
stack ← <x', y'>, lx'y'
lx'y' < 0
次の方向
13: 8方向尽くした場合
14: stack が空 (終わり)
<x, y>, lxy 復元
ENT=2 RETURN: 途中経過印刷用の return
20: pop up
方向の復元
30: lx*y* の復元

付録-1

```
      K=L(M+2,N+2)
      L(M+2,N+2)=K+1
      IF( K.EQ.1 ) IS=IS-1
   31 CONTINUE
      GO TO 15
   32 IF( SP.NE.MAX-1 .OR. NF.NE.1 ) GO TO 30
      B(X,Y)=MAX
      SC(SP)=SC(SP)+1
      C=C+1
      ENT=3
      RETURN
   33 ENT=4
      RETURN
      END
```

条件(2-1)の判定 ⟶

↓解に到達

解の印刷用の return

すべての場合の尽くしおわり